

The Official **AMOS** CLUB

NEWSLETTER ISSUE 5/6

Welcome to the largest AMOS Newsletter yet ! To catch up on the release of newsletters and so that there's room to contain all the new product information, programming and AMOS news, issues 5 & 6 have been combined into one large double issue !

We've got all the regular articles (in double doses) and a new series for the machine coders on how to write AMOS Extensions so that you can add your own commands to AMOS. Not to mention news of the new version of Sprite X and some new AMOS utilities being worked on at the moment.

■ Amiga 500 Plus machines

Despite rumor mongering by some magazine writers, AMOS (and AMOS written) programs written with AMOS 1.3 and above work perfectly on the new A500 Plus computer. There are some problems with 1 or 2 Very old V1.1 RAMOS programs, but most of these have been updated to 1.3 anyway. AMOS also works perfectly (although very fast) on the A3000 machines .

****STOP PRESS**** AMOS 3D will not work with the new A500's. If you have 3D and an A500+, Contact Europress !

All the other AMOS extensions (TOME, Compiler etc) work perfectly.

■ New Sprite X Editor

A new version of Sprite X was released on the 1st of November. It has a host of extra functions, a completely new grabber and animator sections. Not to mention a built in Manual. See Page 13 for information.

■ TOME Goodies Disk 1

The AMOS TOME Goodies Disk is now out. It includes 3 different games written using TOME (including the sourcecode for the popular Magic Forest II), as well as the new SHUFFLE extension, which is listed in this newsletter. The disk is available to registered AMOS TOME Users for £5 .

If you haven't got AMOS TOME yet, then take a peek at the resubscription offer on the back page !

■ Dump Extension

Another new set of commands for AMOS, the Dump extension written by Alex J Grant with modifications by Francois Lionett should be out on public domain soon. It is a simple extension that allows you to do screen dumps to printer from within AMOS using a single command. Should be interesting !

■ New AMOS BBS

Another AMOS BBS has sprung up. Contacting 0527 61839 between 8pm and 4pm will get you in touch with the EX-HELL AMOS BBS run by John Hanton. This new board is free for uploading and



downloading P.D Software. Users who regularly Upload software will be able to gain access to higher levels of the board. I will be contacting several of the AMOS BBS systems soon and dropping off some new software and saying Hi to everyone.

■ More AMOS Info

3 more Independent AMOS Clubs have also appeared in the last couple of weeks. Deja Vu Software (Incorporating the official AMOS P.D Library) have started up a P.D/Licenseware club. Membership of which will get you special offers and discounts on their Licenseware and P.D Software. Contact Sandra Sharkey on 0942 495261 for Info .

Totally AMOS is a disk based magazine written by Len Tucker. The first issue is P.D and further issues are available on subscription. Totally AMOS has reviews and AMOS tips and info as well as some handy programming articles. Issue zero is available from Sandra at Deja Vu.

All about AMOS is a newsletter edited by Peter Hickman, and includes articles on AMOS 3D, simple assembly language for beginners as well as previews and reviews of AMOS Software. All About AMOS Costs £15 per year for a subscription (£18 Europe and £21 Rest of World). Contact Peter at 36 Cleverly Estate, Wormholt Road, London W12 0LX .

None of these groups will have a technical helpline, but could be handy if you want more AMOS Information as well as your AMOS Club Membership !

■ (Yet) Another AMOS Version

AMOS V1.32 is now out (Update on AMOS P.D APD36 as usual), and although Francois still hasn't fixed all the bugs in the Compiler, AMOS V1.32 is quite usable, with several smaller bugs fixed (Such as Music Banks can now be saved with your program and still be recognised by AMOS).

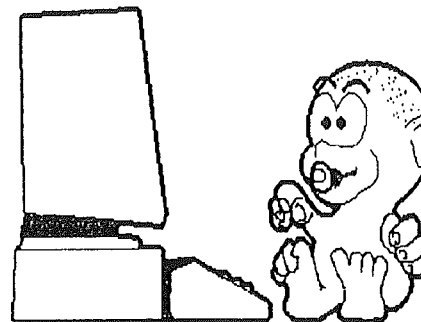
■ Resubscription

Unfortunately it's that time of year ! All subscriptions to the AMOS Club Newsletter expire with this issue's release. Anyone who joined after August can still use the helpline until June (As they won't have had much opportunity yet). To resubscribe to the AMOS Club so that you can continue to receive the AMOS Newsletter, access to the AMOS Helpline and exclusive special offers on AMOS Software (Some only available from the AMOS Club), complete the resubscription form on the back page of this issue.

Inside This Issue:

Absolute Beginners (Disk commands, Variables and Animation).....	3
Program Checker.....	9
Mini Maths 6 (Ping Pong).....	10.
Sprite X V1.32 Review.....	13.
CText Multiple Fonts.....	14.
Shuffle Extension (and how to write your own).....	15.
Messy AMOS (Using AMOS with MS-DOS disks).....	19.
Dos Calling Again ! (Calling CLI Commands).....	20.
Protect & Survive (Combating Piracy).....	21.
Rainbow Warrior Fixes.....	23.
Ten Liners (Place Invaders, Packer & Ann A Conda).....	24.
Using AMOS TOME.....	28.
De-Coloniser (Un Spaghettl your programs).....	31.
Compiler Bug Report.....	32.
Licenseware Reviews.....	33.
Resubscription Special Offers.....	35.
Resubscription Form.....	36.

Absolute Beginners



This issue, I will be covering a couple of subjects that have been causing plenty of helpline calls over the last couple of months. Namely, Amos/Amiga files, and variables.

■ Files

You may or may not be familiar with the Amiga's system for naming files. If not read on, as I will be explaining how the system works, and what some of the file names mean (you will see several types on your AMOS disks).

For this exercise, boot up AMOS, (A backup, not your master !) and go to direct mode (press the ESC key). We will be making use of several AMOS disk handling commands.

First of all, with your AMOS disk in df0 (unless you are working from hard disk), type **DIR** and press Return.

All Amiga files can have names as long as you want (almost), and should not include spaces in them (this can cause all sorts of problems). Usually, the name will be roughly connected with what the file is. For instance, the **AMOS1_3** file on your AMOS disk (or AMOS1_2) is the actual AMOS interpreter program. You will notice with this program, that there is a similar file named **AMOS1_3.info** in the list also. This file is in fact the icon

information file for the AMOS program, that includes the graphical data for the icon, as well as information on the program. When a .info file is included in a directory for a file, that file will appear as an icon on the workbench.

To the right of the file names, you will probably notice the lengths of the files. This length information can be quite handy for problem solving. For instance, if the length of a file is zero, then that file may well have failed to save, (usually from a lack of disk space), and you will have to save the file again elsewhere (or make room on the disk). You will also notice that the AMOS disk has **Directories**. These are "disks within a disk" and can contain even more files. Directories are not files themselves, but drawers in which more files are kept. If you wanted to look inside a directory, you would have to include its name when you do a DIR command. For instance:
Dir "AMOS_System"

will list everything inside the AMOS_System directory on your AMOS Disk (assuming that it is in the drive).

The file length comes in handy if you have to copy a file from one place to another in AMOS. If you know the file length, it is easy. You can use the following program

L = file length in bytes

Reserve as work 5 , L : Rem this reserves bank 5 to the length of the file

Bload "Filename 1" , start (5) : rem this loads the file into bank 5

Bsave "Filename 2" , start (5) to start (5) + L : Rem this saves the file to its new filename.

This does the same as **COPY Filename1 Filename2** does in the CLI.

Unfortunately, it assumes you know the length of the file. A handy trick to find the length of the file is:

```
F$ = " Filename "
```

```
Open In 1, F$
```

```
L = Lof ( 1 )
```

```
Close 1
```

All this does is opens the file as if it were a sequential file, and uses the **LOF** function (**Length Of File**) to find its length. So now you have your own file copying routine. Try the following program.

```
Line Input "Source Filename " ; S$
```

```
Line Input "Destination Filename " ; D$
```

```
Open In 1, S$
```

```
L = Lof ( 1 )
```

```
Close 1
```

```
Reserve as work 5, L
```

```
Bload S$, start ( 5 )
```

```
Bsave D$, start ( 5 ) to start ( 5 )
```

```
+ L
```

This program will allow you to copy files from one place to another on your Amiga disks. As an example, go back to the editor (ESC again), RUN the program and enter:

```
Source Filename S/Startup-sequence  
Destination Filename startup-sequence
```

This will copy the Startup-sequence file from the S directory to the ROOT directory (The root is the basic directory, outside all the others) of the disk. So if you go to Direct mode again and type DIR you will see the file Startup-sequence listed amongst the others.

One of AMOS' useful commands for disk work is **RENAME**, which allows you to change the name of a file on the disk. As an experiment, we'll rename the Startup-sequence file that we have just copied. Type:

```
Rename "Startup-sequence" to "Test"
```

and AMOS will change the name of the file to Test.

Now we only copied this file here as an experiment, so we could do with deleting it from the root directory. If you type KILL "Test"

and press Return, it will eliminate the file from this part of the disk. (It won't touch the original in the S directory though).

■ Disk Names

One rather nice thing about the Amiga disk system is that it can work to either **Devices** (Such as actual disk drive names, like Df0:) or **Volumes** (Particular disks, such as AMOS:). This is downright handy when programming. For instance, in a file utility program, you might just want to access a particular disk drive, such as Df0:, so you can use the Df0: at the start of the filename, or in the DIR command. E.g DIR "Df0:" will give a list of the files in the internal drive. However, in a game, you normally want to access the disk that the game was on, whatever drive it was loaded from. So if the disk was named GAME: for instance, you could use: DIR "GAME:" to list the disk.

One more handy command is the **DIR\$**= function. This changes all access (unless otherwise stated by including a disk name) to the specified disk. So at the start of your game, you could do: **DIR\$="GAME:"** and from then on, all file access would be done from your game disk, and it wouldn't matter what drive it was in. Whereas doing **DIR\$="Df0:"** would depend on the game disk being run from the internal drive.

■ Variables

Right, on to a different subject, **Variables**. What are they, how do they work, what are they for and what different types are there?

A Variable is a place for storing a value

which can be changed by the program (it is Variable, giving a clue to the name). Each variable is given a name, and can hold one value at a time. Whenever a new value is placed in it, the old one is destroyed.

There are 3 types of variable in AMOS, there are:

Integers: These are the normal variables used by AMOS. They can contain whole numbers only (no fractions).

Floating Point: These can contain whole numbers or fractions, and a floating point variable always has a # (Hash) symbol after its name.

String: String variables hold strings of letters and characters and always have a dollar sign (\$, usually referred to as STRING when used with strings) after the variable name.

■ Variable Names

Amos Variable names can be almost any length, and must start with an alphabetical character (A-Z), they can contain numbers, or the Underline character (Shift Minus). Usually, it is a good idea to restrict the length of your variable names, for two reasons.

1) Long variable names take forever to type in

2) A new command might be added to AMOS which is the same as the start of your variable name, thus making your variable illegal.

I would suggest limiting your variable names to about 8 characters maximum, and not including vowels (A,E,I,O,U) or Y, so that the name becomes a scrunched definition of the variable's job. For instance, a variable to hold a number of a dingo could be called **DINGO**, but if someone added a **DING** command (for a different type of bell for instance), the variable would become illegal. So by

calling the variable **DNGO** it is less likely to be effected and is still recognisable.

Next issue I will be starting AMOS by Numbers, a way of learning to program AMOS that is VERY simple, and that lets you program without having to worry about spelling and syntax mistakes, as they aren't possible!

■ And There's more!

Flip the page for part 2 of this double issue's Absolute Beginners!

THE AMOS HELPLINE

Remember that as an AMOS Club member, you can call on the AMOS Helpline if you get stuck.

You'll need your membership number, so have your card handy (write your membership number on all your newsletters!).

☎0271 23544

The best time to phone is between 2pm and 7pm, as this is when I'm most likely to be in, and awake (most of this newsletter was written at 1 a.m in the morning). It's O.k to call outside these times, but don't expect me to be available.

You can also write in with programming queries, but enclose a stamped self addressed envelope, or you won't get a reply. Due to the number of enquiries we can't reply to them all, (especially when all the resubscriptions are coming in) but I do read them all, and those that I can't reply to will be helped in the newsletter.

As expected, those of you who phoned Europress for technical help while I was away in September were told to phone me for help! Oh well!

More Absolute Beginners

Animation made Easier

Last Issue I showed you how to move bobs around the screen with AMOS, and promised that I would show you how to animate them this issue. So get your AMOS'es running and get ready!

First of all, you will need some Images to animate, so run SpriteX or the Sprite Editor and either draw or steal (from Sprite 600 for instance) 4 frames of any animation. This could be a man walking, a propellor spinning, a mouth talking, whatever, as long as it is made up of 4 Images.

If you are stuck while editing your Images, remember to consult the manual before panicking (The manual for Sprite X is built in, just press the Help key and you can scroll through it).

Once you have created your Images, save them as **"ANIMS.ABK"** on your work disk (You should be working with backups of all your AMOS disks, and a blank disk that you can put all your programs on). Then Quit the sprite editor and clear the memory (With the NEW button in the File menu of the AMOS editor).

One of the nice features of AMOS Sprite banks (like the one you just designed or borrowed), is that once loaded into a program, they will stay there until they or the program are erased. This means that you can load the bank in using direct mode, and not have to worry about loading it in your program. When you save your program, the sprites will be saved with it, so you don't usually have to load them again. So press the Esc key to go to direct mode, and type: **LOAD "ANIMS.ABK"**

This will load in your bank of 4

animations. If you saved the Images onto a different disk, then you can do: **DIR\$="diskname:"**

to change the directory to that disk first. Still in direct mode, type the following:

Curs off : flash off : cls 0 (Press Return)

Get Sprite Palette (Press Return)

Bob 1, 100, 100, 1 (Press Return)

Bob 1, 100, 100, 2 (Press Return)

Bob 1, 100, 100, 3 (Press Return)

Bob 1, 100, 100, 4 (Press Return)

This is doing the programming equivalent of flicking pages of an exercise book with one picture on each page. As you can see, it slowly (one frame for each line you type in) animates through the 4 Images. Note that the Image number is the last parameter in the bob command, and that I used the same bob number every time, so that the bob just changed its Image each time. (See Issue 1 for more details of Bob numbers and Image Numbers).

In a program of course, you can get much more complex, and you can make the frames animate more smoothly. Try the following program for instance. (In the Editor of course):

Curs off : Flash Off : Cls 0 : Get

Sprite Palette

For A = 1 to 10

For F = 1 to 4

Bob 1, 100, 100, F

Wait Vbl

Next F

Next A

On running this program, it will animate through the 4 frames 10 times, very very quickly (the whole lot will run through in under a second!). To slow it down change the Wait Vbl command to Wait 5 or Wait 10 (or Wait 100 if you want slow

mo!).

This is the basis of bob animation, changing a bob's Image each frame, to give the impression that it is changing. You will notice in the above program, that the variable F is used to determine the Image number to be used each frame (F counts from 1 to 4 using the For..To..Next statements). The Image number can be a fixed constant, like

```
Bob 1, 100, 100, 3
```

a variable

```
Bob 1, 100, 100, F
```

A formula

```
Bob 1, 100, 100, 1 + rnd ( 2 ) + F
```

or even a very complex formula like

```
Bob 2 + NB, ACX - MX, ACY - MY,
S ( R / 2 + HVR * ( 16 + GTG ),
Max ( 0, UC - FUC ) ) - 22 *
SWING * ( SP# < 4 or TH < 8 )
```

(A line from Jet Strike, Coming soon!) It is how you make the program run through these animations that is important.

Normally, you would keep a variable which tells the program what Image number is to be used. For instance F. Then whenever you change that variable, the next time the bob is redrawn with BOB n, x, y, F the bob will change Image. Normally a game loop will be set up to carry on going without worrying about the animation frame (as movement tends to be more important), so you have to make sure that your variable changes the way you want it.

For instance, in the following program, the bob will only animate if you are holding the fire button down. Use has been made of the **MOD** (Modulus) function to make the variable F loop around when it gets too high. **MOD** returns the remainder of a division (e.g 5 divided by 2 is 2, remainder 1, so 5 Mod 2 would

return 1).

```
F=0
Do
Bob 1, 100, 100, F + 1
Wait 10
If Fire ( 1 )
F = F + 1 : rem you could also do Inc
F
F = F Mod 4 : rem F will go 0, 1, 2, 3, 0, 1, 2, 3 etc
End If
Loop
```

try this one out. To end it you will need to press Ctrl-C.

A lot of the time, you will want to make a bob animate when you move it. This depends on exactly how it is being moved, and there are several ways of determining the Image number to display.

1) Make it random (rarely used) E.g F = Rnd (3)

2) Calculate it from the co-ordinates of the bob. E.g F = X Mod 4

3) Change it every time the bob is moved E.g If Jleft(1) : X = X - 1 : F = F + 1 : F = F Mod 4 : End If

This method is quite popular. An example of this method follows:

```
F = 0 : X = 160
Do
Bob 1, X, 100, F + 1
Wait 10
If Jleft ( 1 )
X = X - 1
F = F + 1
End If
```

```
If Jright ( 1 )
X = X + 1
F = F + 1
End If
F = F Mod 4 : rem makes sure that F
is from 0 to 3
Loop
```

As you move the bob left or right, the animation will occur.

Ok, you're going to have to design some bobs for the next part. You'll need 8 images as 4 pairs of 2 frames. Each pair should be of a person (or object if you're as bad at graphics as I am) walking in one direction. Two different frames are used for each direction, to give a simple animation as the object moves. Your list of images should look like this:

```
Image
1 Facing Left frame 1
2 Facing Left frame 2
3 Facing Top frame 1
4 Facing Top frame 2
5 Facing Right frame 1
6 Facing Right frame 2
7 Facing Bottom frame 1
8 Facing Bottom frame 2
```

Once you have done these (in Sprite X or the Sprite Editor), save them as a sprite bank and clear the program. You'll notice that we now have 4 rotations of the object/person in pairs, this means that if you used a variable for the rotation (say R for instance), you could work out the image number for the first frame of that rotation by $\text{frame} = R * 2 + 1$, so the command `Bob 1, x, y, R * 2 + 1` would display the first frame of rotation R at the co-ordinates x,y.

The following program shows how you can control this variable with the joystick while moving the object around:

```
Load "whatever your bank is called .
abk"
X = 160
Y = 100
Curs Off : Flash Off : Cls 0
Get Sprite Palette
R = 0
```

```
Do
Bob 1, X, Y, R * 2 + 1
DX = 0 : DY = 0
If Jleft ( 1 )
DX = - 1 : DY = 0
R = 0 : rem face left
End If
If Jright ( 1 )
DX = 1 : DY = 0
R = 2 : rem face right
End If
If Jup ( 1 )
DX = 0 : DY = - 1
R = 1 : rem face up
End If
If Jdown ( 1 )
DX = 0 : DY = 1
R = 3 : rem face down
End If
X = X + DX
Y = Y + DY
Loop
```

If you run this program, the object will move around the screen facing in the direction of movement. It won't animate yet however, because we are only using the first frame of the animation. To make the object animate as it moves, try the following

1) Insert `T=0` as a line between the `R=0` and `Do` lines

2) Change the `Bob 1,X,Y,R*2+1` line to `Bob 1,X,Y,R*2+T`

3) add the following lines in between the `Y=Y+DY` and `Loop` lines:

```
If DX<>0 or DY<>0
T=1-T
EndIf
```

and run the program. The `T=1-T` bit makes T change from 0 to 1 and vice versa if the bob is moving (i.e if DX is not zero, or DY is not zero). Just before I go, if you want to make the object move continually once moved, remove the `DX=0 : DY=0` line!

PROGRAM CHECKER

One of the hazards of typing in a listing from a magazine is that typing errors very often creep in. Either at this end, or when you are entering the program. So several members have asked about a checksum system for the newsletter, where a checksum (a special code worked out from the contents of each line) is printed along with each line.

When you have entered your program, you can save it out as an ASCII file, and run it through the checksum program to make sure that the checksums for each line are the same as those in the newsletter. If a line's checksum is different, then you (or sometimes we) have made a typing mistake on that line. So you know which line will cause the program to fail.

The following program will generate a checksummed listing, and can also be used to check listings that you have typed in. It has been run on itself, so that when you type it in, you can run it (assuming you typed it in correctly) and check it. To check a program (and to create a checksum listing of it), simply save it as an ASCII file when you have finished typing it in.

To save a program as an ASCII file, do the following:

1) press Ctrl and A at the same time. This will mark your whole program as a block.

2) Click on the Blocks menu button (at the top of the editor screen).

3) Click on Save ASCII on the menu. A file selector will now appear, simply enter the name of the program with `.ASC`

after it, and it will be saved as an ASCII file. Once this is done, load in the Checksum Checker, and run it.

A file selector will let you choose which program you want to check, and another file selector will let you pick what file to save the Checksum listed program as. If you don't need a listing of the program with the checksums after the lines, simply enter `TEST.ASC` as the file name on this second file selector. The program will then load in your listing and calculate the checksum for each line (Spacing is ignored in the lines) and will display the checksums for the lines on the screen. All you need to do is make sure that the checksums on the screen are the same as listed in the newsletter. If they are not, then there is a mistake on that line of your listing.

```
1) ' Program Checksum Generator [ 45]
2) ' By Aaron Fothergill [ 99]
3) ' AMOS Club 1991 [ 47]
4) Rem Calculates Checksums for
   program lines [ 65]
5) Screen Open 0, 640, 200, 2,
   Hires : Curs Off : Flash Off : Cls 0
   [ 78]
6) Colour 1, $FFF [ 119]
7) Set Input 10, - 1 [ 37]
8) F$ = Fsel$ ( " * . ASC ", "", "
   Load an ASCII AMOS program " )
   [ 47]
9) If F$ <> "" [ 111]
10) If Exist ( F$ ) [ 31]
11) Open In 1, F$ [ 76]
12) OK = 1 [ 40]
13) End If [ 96]
14) End If [ 64]
15) F2$ = Fsel$ ( " * . ASC ", "",
   "Save it as ?" ) [ 48]
16) If F2$ <> "" [ 93]
17) Open Out 2, F2$ [ 52]
18) OK2 = 1 [ 26]
19) End If [ 64]
```

```

20) L = 0 [ 65]
21) Paper 0 : Pen 1 [ 54]
22) Print "Press any key to Pause"
    [ 94]
23) While OK = 1 and OK2 = 1 [ 38]
24) If Inkey$ < > "" : Clear Key : Wait
    Key : End If [ 113]
25) Inc L [ 40]
26) E = Eof ( 1 ) [ 36]
27) If E = 0 [ 103]
28) Line Input # 1 , L$ [ 46]
29) Gosub CHECKSUM [ 97]
30) Else [ 63]
31) OK = 0 [ 41]
32) End If [ 96]
33) Wend [ 24]
34) Close 2 : Close 1 [ 57]
35) Print "Finished" [ 67]
36) End [ 79]
37) CHECKSUM : [ 55]
38) L2$ = L$ [ 15]
39) Rem remove spaces [ 75]
40) For A = 1 To Len ( L$ ) [ 35]
41) If Mid$ ( L$ , A , 1 ) = " " [ 111]
    (That's a Quote Space Quote)
42) L$ = Mid$ ( L$ , 1 , A - 1 ) + Mid$
    ( L$ , A + 1 ) [ 69]
43) End If [ 96]
44) Next A [ 70]
45) L$ = Upper$ ( L$ ) [ 90]
46) C = 0 [ 78]
47) For A = 1 To Len ( L$ ) [ 35]
48) C = C xor Asc( Mid$ ( L$ , A , 1
    ) ) [ 117]
49) Next A [ 70]
50) Print " ( " ; L ; " ) =" ; C [ 89]
51) Print # 2 , Str$ ( L ) + " ) " +
    L2$ + " [ " + Str$ ( C ) + " ] "
    [ 61]
52) Return [ 10]

```

Now you've typed it in, save it as a .AMOS program, and then as a .ASC program so that you can use it to check itself!

■ Mini Maths 6

Ping Ponging

Nope, not another 10-liner game, but another version of a toggle, where a variable progresses from one value to another and then back again, rather like a ping pong ball going from one side of a table to another.

This is achieved by having one variable for the value, and another for the vector. When the value reaches either limit, you reverse the vector.

Have a go with the following example.

Rem Ping Pong Example

X = 1 : DX = 1

Curs off : Flash Off : Cls 0

Ink 4 , 4

Bar 0 , 0 to 15 , 15

Get Bob 1 , 0 , 0 to 16 , 16

Double Buffer

Do

Bob 1 , X , 100 , 1

Wait Vbl

X = X + DX

If X <= 0 or X >= 319

DX = - DX

Endif

Loop

CText Kerning

The AMOS 1.3 version of CTEXT has the **KERN\$(n)** function built in.

So to make a character kern, you simply add **KERN\$(kern)** into the string you are printing, with kern being the number of pixels left or right that you want to kern.

If you are using CTEXT, then read page 14 for information on using multiple fonts!



Now being used by **ALL the major AMOS Programmers**, including Aaron Fothergill, Len Tucker (Sparx), Gary Symons, Stephen Hill, Peter Hickman and more, **AMOS TOME** has been upgraded to **V3.1** to be compatible with the AMOS Compiler and AMOS V1.3. It now has **34 commands**, and the editor is **still the most powerful map editor available on ANY computer**.

TOME has already been used to write 5 commercial games, and many more are being developed.

90% of games use map based backgrounds, making **AMOS TOME the most useful extension for AMOS**.

The TOME Goodies disk has been released (available only to registered TOME users) including a **high speed scrolling 3D isometric game, 3D dungeon game and Magic Forest II**, a horizontally scrolling platform game, as well as an extra manual for AMOS TOME programmers wanting to use these routines in their own games.

AMOS TOME costs **£19.99** for AMOS Club members (**£24.99** for non members) and is only available direct from Shadow Software in the U.K.

When ordering AMOS TOME, please make cheques payable to Shadow Software and add **£5** postage & packing if ordering from outside the U.K.

Send your order to:

Shadow Software, 1 Lower Moor, Whiddon Valley, Barnstaple, N.Devon. EX32 8NW.

All U.K. prices are inclusive of V.A.T at 17.5% (So you don't have to add any more!)

Please send me AMOS TOME (V3.1).....

Name

Address

Postcode.....Membership Number.....

I enclose a Cheque/Postal Order for **£19.99** (members only), **£24.99** (Non-Members) + **£5** P&P if ordering from Overseas. (P&P is free in the U.K.)

AMOS CTEXT

A brand new AMOS Extension, 5 new commands, gives you easy to use Icon based text display. Ctext can do:

Fonts of up to 64 colours,

Any size of character,

Easily edited fonts (Can be edited in SpriteX)

Masked fonts,

Proportional text,

Kerning (In the 1.3 version)

Ctext is supplied in AMOS 1.23, 1.3 and ASCII versions with a font setter program, demonstrations and example P.D fonts.

It is **available direct from the AMOS Clubmembers**, for £2.50 U.K, £3.50 Overseas : (But take a look at the resubscription offer first !)

Order Form

Please send me:

AMOS SPRITEX V1.32 (ACLUB 1) ☐ £2.50 (£3.50 R.O.W)

AMOS CTEXT (ACLUB 2) ☐ £2.50 (£3.50 R.O.W)

Name

Address

Postcode Membership Number

I enclose a cheque/Postal Order for £..... (please make payable to Shadow Software)

AMOS SpriteX v1.32

The enhanced version of the AMOS Sprite Editor, the latest version now includes:

Auto scrunch (automatically removes waste area from sprites)

Can edit Sprites, Bobs and Icons directly (including the correct palettes)

Includes new improved built in bob grabber and animator.

Automatic NTSC detection, Full 10th of a Degree Rotation.

Now supplied as AMOS 1.23 and 1.3 versions

SpriteX costs **£2.50** (£3.50 Overseas) when ordered **Direct from The AMOS Club** with the above order form (but look at the resubscription offer first !)

SPRITE X EDITOR

A new version of the popular AMOS Sprite X enhanced sprite editor will be released on November 1st. This version has several new functions, and many of the older ones have been enhanced.

This version will be released as Licenseware (£3.50 via Deja Vu or Licensed Distributors), but AMOS Club members can order it **DIRECT FROM THE AMOS CLUB**, for £2.50. AMOS Ctext will also be released into Licenseware, and can also be ordered direct from the club for £2.50.

Sprite X's new features are: Full rotation, with accuracy down to 10th of a degree.

Can automatically generate rotations into the bank.

Handles both bobs and icons, including CTEXT font icons.

Built In Grabber X sprite grabber and ANIM X animation tester. Grabber X can grab individual sprites from a picture, or a whole line of them at once!

ANIM X can animate up to 16 bobs over 51 frames, over an IFF background screen, so that you can test out your animations for your game without having to leave Sprite X.

Built In Manual. Pressing the Help key loads in the manual, which can be scrolled through within the editor (AMOS 1.3 version only).

Other features (such as the UNDO button) have been updated and modified.

There are now several versions of the Sprite Editor and Sprite X available, and to avoid confusion of the "what version have I got?", here's a list:

Sprite Editor: Provided with AMOS. Modified to work with AMOS 1.3 in later

update disks, can be spotted by the Blue, Red and Yellow Icons, and old style AMOS logo bar (No Load or Save Icons).

Sprite X V1.0: Early Limited Licenseware version, also provided free with AMOS Compiler, 3D and some update disks. New style icons, included Load, Save, Append and simple sprite grabber and animator.

Sprite X V1.1: Updated version available on Limited Licenseware, included 10th of a degree rotation, and automatic scrunch facility.

Sprite X V1.32: The new version, built in manual, new Grabber X and Anim X built in utilities.

After November 1st, you can update your old version of Sprite X to Sprite X V1.32 for just 72p + 24p postage by sending your **original AMOS CLUB Licenseware disk** (Not the Compiler, 3D or update disk versions of SpriteX) to the **AMOS Club**, along with 4 1st class stamps (to cover the cost of the new disk, and return postage). The old disk **must be the ACLUB 1 Licenseware version of SpriteX on its original disk**, not the free version given away with the Compiler, 3D or Update disks. This offer only applies to AMOS Club members, so include your membership number!

As mentioned above, we have arranged with Deja Vu, so that all Licenseware developed by us for the AMOS Club will go onto full licenseware after being available only to AMOS Club members for a few months (which gives you a couple of months advantage on non members), and once released on Licenseware will be available direct from the AMOS Club for £2.50 rather than the normal £3.50 per disk. This only applies to AMOS Club limited Licenseware, such as Sprite X and CTEXT, and doesn't apply to the other licenseware disks, which you still have to buy from Deja Vu, or one of the Licensed Distributors.

CTEXT Multiple Fonts

Since CTEXT was released just over 2 months ago, it has been rapidly replacing the use of IFF fonts in most programmers AMOS programs. I have had several questions about how to load in more than one CTEXT font at a time and switch them about when required. This problem has been solved in two ways.

Firstly, the following program demonstrates how to load in multiple CTEXT fonts, and secondly, I am adding to the CTEXT extension, so that it can handle multiple fonts in a much easier, memory saving way. This version should be ready for the new year, at which time an upgrade will be available.

- 1) Rem multiple CTEXT fonts demonstrator [68]
- 2) Rem by Aaron Fothergill . The AMOS Club [65]
- 3) Rem [90]
- 4) Dim FT\$ (5) : Rem array to store font names [120]
- 5) Screen Open 0 , 320 , 200 , 16 , Lowres : Curs Off : Flash Off : Cls 0 [45]
- 6) NFNT = 3 : Rem number of fonts to load [43]
- 7) For A = 1 To NFNT [31]
- 8) Read FT\$ (A - 1) [88]
- 9) Next A [70]
- 10) Data " futuresmall . abk " , " futurebig . abk " , " andover . abk " [32]
- 11) Reserve As Work 6 , 768 * NFNT : Rem reserve space for font data [118]
- 12) Rem load in the fonts [66]

- 13) For A = 0 To NFNT - 1 [2]
- 14) Load "CTEXT : FONTS/" + FT\$ (A) [112]
- 15) Get Icon Palette [32]
- 16) Make Icon Mask : Rem for masked font [0]
- 17) Bank Swap 2 , 11 + A : Rem swap it to spare bank [103]
- 18) Bload "CTEXT : FONTS/" + FT\$ (A) + " . CFNT " , Start (6) + A * 768 [10]
- 19) Next A [70]
- 20) Rem swap the first one into the font bank [88]
- 21) Bank Swap 11 , 10 : LBANK = 11 [115]
- 22) Font Size 0 , 0 : Paper 0 : Rem proportional text [10]
- 23) Do [11]
- 24) Cls : F = - 1 [33]
- 25) While F < 1 or F > NFNT [67]
- 26) Input "Font Number" : F [27]
- 27) Wend : Dec F [6]
- 28) Copy Start (6) + F * 768 , Start (6) + 768 + F * 768 To Font Data [3]
- 29) Bank Swap 10 , LBANK [84]
- 30) LBANK = 11 + F [58]
- 31) Bank Swap 10 , LBANK [84]
- 32) Line Input "Enter Text:" : T\$ [92]
- 33) Cls 0 : Ctext 10 , 100 , T\$ [104]
- 34) Wait Key [124]
- 35) Loop [28]

Remember not to include the line numbers before the round brackets, or the checksums in the square brackets at the end of the lines. These are so that you can run the program through the Checksum Checker in this issue.

CText is now available to AMOS Club members direct from the AMOS Club for £2.50, or from Deja Vu Licensed Software for £3.50 and take a look at the special offers when you resubscribe!

THE AMOS SHUFFLE EXTENSION

This is one for the machine coders amongst you (For non coders, either get a coding friend to assemble it for you or the Shuffle extension is on the TOME Goodies Disk 1, ready to go). This version is for AMOS 1.3 and above, using the new extension system. If you have an earlier version of AMOS, upgrade now, as you'll be missing out!

The Shuffle extension contains 3 commands to make life a little easier, it is assembled as AMOS extension number 9 (Which is now reserved for it, and future experimental extensions). The 3 commands are:

= Range (a , b to c) : Returns value a if it is between b & c, if it is lower than b, it returns b, and if it is higher than c, it returns c. Effectively the same as = Max (b , Min (c , a))

Shuffle bank , n , seed : Shuffles the numbers 1-n, into reserved bank, using seed as a random seed. Useful for card games, bingo games and appears, as it can shuffle 10,000 numbers in as little as 4 frames (4/50ths of a second)!

= RAND (bank , n) : returns the nth number from a list of previously shuffled numbers. = RAND (bank , n) can also be used to retrieve numbers directly from any bank, as it does the same as = Deek (Start (bank) + n * 2).

To use shuffle, you must first enter the program in your assembler (listed here for Devpac 2), the @ symbol is used here to represent pressing the TAB key. Once you have assembled it (it should assemble as position independent), copy the resulting file (call it **RANGE.LIB**) into the AMOS_System directory, boot up AMOS, and load in the **Config1.3.amos** program. Load in the default Configuration,

and go to the LOADED EXTENSIONS page. Then change the line for extension 9 to read

: AMOS_System / RANGE . Lib

Save the configuration, wait for the disk to stop working, and then re-boot AMOS. You should now have the AMOS Club Shuffle extension installed.

To use the Shuffle command, you must do the following:

1) Reserve a bank (Work) to the number of numbers you want to shuffle, times 4 . e.g

N = 52 : rem pack of cards
Reserve as work 5 , N * 4

2) Use the Shuffle b,n,s command e.g
Shuffle 5 , N , Timer

If you use the same number as a seed, then the routine will return the numbers in exactly the same order, so it is a good idea to use Timer, or some other variable amount as the seed.

3) This will fill bank 5 with the numbers from 1 - N, shuffled into a random order, with no repeats. You can now use the = RAND (bank , n) function to pull them out in any order: e.g

For A = 1 To N
Print Rand (5 , A)
Next A

This will print out all the numbers in the order they were shuffled.

The listing contains comments on how the extension is written, so that you machine coders can use it to write your own extensions. Extension 9 is now reserved as an experimental extension, so that you can use it, as long as you don't expect it to be the same on a different machine. This is so that there is always an extension number available to try out different extensions before they are made ready for release.

- * AMOS Club Extension
- * Extension number 9 for AMOS
- * By Aaron Fothergill
- * Thanks to Francois for the notes on extensions.
- * Includes the =RANGE(a,b to c) function
- * And the SHUFFLE bank,n,seed command
- * as well as the =RAND(bank,x) function

```
ExtNb @ @ equ @ 9-1 @ ; extension
number - 1
```

```
@ @ Include @ "_Equ.s"
@ @ RsSet @ DataLong
@ @ Include @ "_Pointe.s"
@ @ Include @ "_WEqu.s"
@ @ Include @ "_CEqu.s"
@ @ Include @ "_LEqu.s"
```

- * First we have a series of lengths of the routines and tables

```
Start
@ @ dc.l @ X_Tk-X_Off @ ; length of
offsets
@ @ dc.l @ X_Lib-X_Tk @ ; length of
tokens
@ @ dc.l @ X_Title-X_Lib ; length of
libraries
@ @ dc.l @ X_End-X_Title ; length of
Title string
@ @ dc.w @ 0
```

```
*****
* @ @ OFFSETS TO FUNCTIONS
X_Off @ dc.w ( L1 - L0 ) / 2 , ( L2 -
L1 ) / 2 , ( L3 - L2 ) / 2 , ( L4 - L3
) / 2
@ dc.w ( L5 - L4 ) / 2 , ( L6 - L5 ) /
2 , ( L7 - L6 ) / 2 , ( L8 - L7 ) / 2
* always have a few spare in case
you want to add to your extension
*
* @ @ Token table
X_Tk @ @ dc.w @ 1,0
```

```
@ @ dc.b @ $80,-1
* always include the above 2 lines
@ @ dc.w @ - 1 , L_Range
* The above two words are offsets
to the library for the first
* command. The first offset is to the
COMMAND version, the second
* to the FUNCTION. If one isn't
required, then you use a -1. In the
* Range function, there is no
COMMAND version, so the -1 is placed
as
* The first word.
@ @ dc.b @ "rang" , "e" + $80 , "00
, 0 t 0" , - 1
* The first string contains the
command/function name
* The last letter of which has the hi
bit set (thus the "e"+$80)
* The second string determines the
parameters.
* For the Parameter strings, the first
Character determines the type
* 1 = an instruction (no return
parameter)
* 0 = an integer is to be returned
* 1 = a floating point number is to be
returned
* 2 = a string is to be returned
* V = it is a reserved variable
* Then the rest of the numbers tell
AMOS what parameters are
* needed by the command/function.
* These can be separated by commas,
or a t (for TO) and can be
* either 0,1 or 2 for the 3 variable
types.
* In this case (a,b to c) returning an
integer (The 0 at the start)
@ @ dc.w @ L_Shuffle , -1
@ @ dc.b @ "shuffl" , "e" + $80 , " 1
0 , 0 , 0 " , - 1
@ @ dc.w @ - 1 , L_Rand
@ @ dc.b @ "ran" , "d" + $80 , " 0
0 , 0 " , - 1
@ @ dc.w @ 0
```

- * always end the token list with a 0

```
*****
* @ @ Start of library
X_Lib:
*****
* @ @ COLD START
* The first library routine is used to
set up any parameters needed
* by the extension. In this case, we
just need to tell AMOS the
* extension number (by passing it in
Dreg 0)
L0
@ moveq @ #ExtNb , d0
@ rts
*****
L1
* A couple of spare routines always
comes in handy
*****
L2
*
L3
* =RANGE(a,b TO c)
* return result of a forced into range
of b to c
* shorter version of =MAX(b,MIN(c,a))
L_Range @ equ @ 3
@ move.l @ (a3) + , d2 @ ; get the
last parameter
* The above pulled an integer from the
stack. (as required)
@ move.l @ (a3) + , d1
@ move.l @ (a3) + , d0
* note that they are pulled in reverse
order
@ @ cmp.l @ d2 , d0 @ ; is a>c ?
@ bgt @ BIGGER ; if so then force
a=c
Bt: @ cmp.l @ d1 , d0 ; is a<b ?
@ bit @ SMALLER @ ; if so then
force a=b
* when you have to return a
parameter from a function,
```

- * Set D2 to the type (0,1 or 2) and D3 to the value (or
- * the address in the case of a string)

```
B2: @ move.l @ d0 , d3
@ move.l @ #0 , d2
@ rts
BIGGER:
@ move.l @ d2 , d0
@ bra @ B1
SMALLER:
@ move.l @ d1 , d0
@ bra @ B2
L4
* Shuffle bank,number,seed
L_Shuffle @ equ @ 4
@ move.l @ (a3) + , d0 @ ; random
seed
@ move.l @ (a3) + , d1 @ ; Number of
numbers
@ move.l @ (a3) + , d2 @ ; bank
number
@ Rbsr @ L_Getbank @ ; turn bank
number into address
* Note the use of Rbsr, this is a
macro (In the included equates files)
* used for jumping between libraries
instead of bsr. Francois has done
* R macros for all the normal jumps.
@ move.l @ #0 , d3 @ ; counter A
@ move.l @ #1 , d4 @ ; counter B
IL:
@ move.w @ d4 , ( a2 , d3.l )
@ addq.l @ #2 , d3
@ addq.l @ #1 , d4
@ cmp.l @ d1 , d4
@ blé @ IL
@ move.l @ d1 , d3 @ ; d3= numbers
remaining
@ move.l @ d1 , d4 @ ; pointer to
second part of bank
@ asl.l @ d4 @ ; x2 so it points to
the second half
@ move.l @ d1 , d5 @ ; A (Counter
down to 0)
@ subq.l @ #1 , d5
@ move.l @ d0 , d2 @ ; Pointer.
```

```

Starts off as seed \
SL:
MIB: @ cmp.l @ d3 , d2 @ ; check
      that P<L
@ bge @ MOD1
@ move.l @ d2 , d6 @ ; d6=pointer x
  2
@ asl @ d6
@ move.w @ 0 ( a2 , d6.l ) , d7 @ ;
  get number
@ move.w @ d7 , 0 ( a2 , d4 ) @ ;
  put it in second list
@ addq.l @ #2 , d4
@ move.l @ d3 , d7
@ subq.l @ #1 , d7
@ asl.l @ d7 @ ; Last number
@ move.w @ 0 ( a2 , d7.l ) , a0
@ move.w @ a0 , 0 ( a2 , d6.l ) @ ;
  swap it with number taken
@ sub.l @ #1 , d3 @ ; L=L-1
@ add.l @ d0 , d2 @ ; P=P+seed
@ add.l @ d2 , d0 @ ; seed=seed+p
@ and.l @ d3 , d0 @ ; seed=seed and
  L
@ sub.l @ #1 , d5
@ cmp.l @ #1 , d5
@ bne @ SL
@ move.w @ 2 ( a2 ) , d7
@ move.w @ d7 , 0 ( a2 , d4 )
@ addq.l @ #2 , d4
@ move.w @ 0 ( a2 ) , d7
@ move.w @ d7 , 0 ( a2 , d4 )
* Move block to start of bank
@ clr.l @ d0
@ move.l @ d1 , d2
@ asl.l @ d2
@ move.l @ d2 , d3
EL: @ move.w @ 0 ( a2 , d2 ) , d4
@ move.w @ d4 , 0 ( a2 , d0 )
@ addq.l @ #2 , d0
@ addq.l @ #2 , d2
@ cmp.l @ d3 , d0
@ bne @ EL
@ rts
MOD1: @ sub.l @ d3 , d2

```

```

@ bra @ MIB
*
L5
L_Rand @ equ @ 5
* =RAND(bank,n)
* Pulls a number from a shuffled stack
*
@ move.l @ ( a3 ) + , d0 @ ;n
@ move.l @ (a3) + , d2 @ ; bank
  number
@ Rbsr @ L_Getbank
@ asl.l @ d0 @ ; x2
@ clr.l @ d3 @ ; value is returned in
  D3
@ move.w @ 0 ( a2 , d0 ) , d3
@ clr.l @ d2 @ ; D2 is set to 0, to
  show
@ @ @ @ ; that it is integer
@ rts

L6
*
L7
* Bank number in d2, returns bank
  address in a2
L_Getbank @ equ @ 7
@ move.l @ ABanks( a5 ) , a0
@ tst.l @ d2 @ ; check bank number
@ beq @ AdBE1 @ ; too low (0)
@ cmp.l @ #16 , d2 @ ; too high ?
@ bge @ AdBE1
@ subq.l @ #1 , d2
@ asl.l @ #3 , d2 @ ; multiply by 8
@ move.l @ 0 ( a0 , d2 ) , d3 @ ;
  Get bank address
@ tst.l @ d3
@ beq.s @ AdBE1 @ ; if zero, bank
  not reserved
@ move.l @ d3 , a2
@ addq.l @ #8 , a2
@ rts
AdBE1: @ moveq @ #36 , d0
@ Rjmp @ L_Error
*
L8
* @ @ Welcome message

```

```

X_Title
@ dc.b @ 31,"Shuffle Extension" , 0
@ even
*****
X_End @ dc.w @ 0
* end the extension with a 0

```

Messy AMOS

By Jeff Tullin

If you want to read & write IBM discs in your Amiga drive from an AMOS program, there are 2 ways. Either you use Cross-Dos, the commercial file transfer package, or you use Messydos which is available from most P.D Libraries as shareware.

Messydos however, is not an easy program to use in the form it appears in the P.D Libraries, and most people just give up with it rather quickly. I have just set up my AMOS disk to use it, however, and I'd like to tell you how it's done, should you want to do the same.

1) Buy a copy of Messy Sld II (See Below), you could use the original Messy-Dos disk, but it's much easier this way.

2) Backup your AMOS program disk.

3) Copy the following files from Messy-Sld II to AMOS:

```

MessySld I I : Devs / messydisk.device
to AMOS : devs
MessySld I I : Devs / mountlist to
AMOS : devs
MessySld I I : L / Messyfilesystem to
AMOS : L

```

4) Now, all that remains is to change your startup sequence to read:
Mount MS0: (or MS1 if you have an external drive)
Run >NIL: AMOS1 . 3
Endcll

Now when you boot up this AMOS disk, you can access an MS-DOS disk by using the device MS0: (or MS1:) instead of DF0: The good thing about this system is that you can read both MS-DOS disks and Amiga Dos disks in the same drive.

Messy Sld II is available from Deja Vu P.D and N.B.S, and is Shareware. If you find yourself using Messy Dos in your programming, then please make sure you send off the shareware fee to the author, as it would be unfair to do otherwise, and is more likely to persuade him to write another version.

■ Notes on Shareware

Remember that a Shareware author is trusting you to pay for a program if you think it is worth it. In the case of Messy Dos, the author asks you to send whatever you think is suitable (I would suggest at least £10 if you are going to use the program, much more if you make money from anything written using it!)

If no-one pays for Shareware, then it will fade away, and the gap between writing simple programs for nothing, and complex programs for great rewards will widen, making it even more difficult for good programmers to stay in business.

Digitizing Service

Technical Fred are now able to print T-Shirt transfers (in colour) as well as digitising still pictures and audio.

Phone them on **061 703 7842** for details.

D O S C A L L I N G

AGAIN!

■ Calling a CLI
command from within
AMOS

The following routine was supplied by Rod Pascoe, and was first published in Totally AMOS (Len Tucker's disk based magazine) in September, unfortunately though, a typing error crept in and that version didn't work. Here's the working version. This will only work with AMOS V1.23 and above (as you need the AMOS TO BACK/FRONT commands).

```
Amos To Back
Rem Hides AMOS
COM$ = " openWb "
Rem COM$=the CLI Command you wish
to call
Dreg ( 1 ) = Varptr ( COM$ )
Rem D1 contains the pointer to the
command string
Dreg ( 2 ) = 0
Rem D2 contains the Input Channel
(0=default)
Dreg ( 3 ) = 0
Rem D3 contains the output Channel
(0=default)
A = Doscall ( -222 )
Rem call the Execute routine
Amos To Front
Rem bring AMOS Back
```

The CLI Command will now be activated, and will return to AMOS when finished. The command you are calling must be in the C directory of your boot disk, and a CLI or shell window must be open for this routine to work.

As the routine uses a DOS Call, which can Guru if you don't get the parameters correct, it isn't for beginners!

By the way, you can toggle between the CLI program and AMOS with Left Amiga-A.

Thanks again to Rod, who is compiling a list of useful Dos & Exec call commands for the newsletter.

And thanks also to Len Tucker, for the Totally AMOS disk magazine (You'll find my Logger Larry 10 - liner Donkey Kong Clone on his Issue 1 disk).

The Game Maker's Manual, Preview

I have recently been talking to Stephen Hill, author of the AMOS Manual, about his forthcoming book, the Game Maker's Manual (to be published by Sigma Press, at around £20).

The text is now finished, and once proof read and printed, it should be available just after Christmas.

The book is written to show how to write various types of games with AMOS, and assumes that the reader can program in BASIC, but doesn't know any of AMOS' extra commands.

It covers subjects such as Adventure games, arcade games, Trader games and Map based games, and also includes chapters on the AMOS Compiler and AMOS TOME, as well as using music and sound.

All the subjects include programming examples, and some include full games for you to type in (and take apart).

We'll have a full review once copies are available, this one looks like being a good investment for anyone trying to program their first game in AMOS!

Protect & Survive

Professor Speck's guide to combating piracy

Many things have been said on the subject of software piracy, how it is only done because software costs too much, because it's a challenge and so on. I could explain a few techniques of how to make your disks piracy protected, but this wouldn't be much use, as it is possible to pirate ANY software on the Amiga. By copy protecting your disks, you are merely giving in to the software terrorists, putting your own workload up and wasting effort.

Thus, the Professor Speck Protect and Survive guide, of how to spot and eliminate software piracy. Remember that piracy is killing off the home computer.

Since the A500 some years ago, there has not been a major home computer release, and both Atari and Commodore are working on CD or Cartridge based machines, with nothing new lined up for home use (unless you can afford a TT or A3000). This article is intended to give you a guide to how you can spot and combat piracy in a **legal and non-violent** way. You should **NEVER** break the law or use violence against anyone, it is the worst thing you can do, makes you as bad as them and usually gets you into more trouble than the pirate. The basic guideline is "Let's be careful out there!"

■ Spotting Piracy

A typical software pirate will be roughly humanoid and look quite normal. There is no stereotyped software pirate.

This means that you have to spot a pirate by other means. Obvious signs of piracy are software in photocopied packaging sold at very cheap prices on markets, from a very portable suitcase for instance. Another good place to look

is computer clubs. Every computer club I have been to has usually had at least one group in a corner, doing nothing but pirating software. This is usual in school or college computer clubs, even teachers have been known to break the law by copying software. You can usually spot that someone is pirating, if they seem to have a lot of boxes of disks, with very little public domain software. I have known several pirates, who simply collected software, and never played any of it.

Also watch out for someone who orders disks in large bulk, but isn't running a software company or P.D library. Magazines have also been known to break copyright, and there have been several cases of magazines using software from public domain libraries without checking the READ.ME files. A lot of Public Domain software (most of it) is put on P.D with the condition that no profit is made on it. Which is something that magazines with cover disks make a lot of. Some magazines have also put programs on their cover disks without checking to see if they were actually P.D first. And have you noticed, that several magazines use the Slipstream loader program, which is advertising one of the major pirate groups!

The hardest pirate to spot, hardest to eliminate and the one that is causing the most damage, is the "copying for friends" idiot. This is the person who will casually break copyright and copy software for his friends, expecting them to do the same for him.

■ How to Stop it

Piracy is very difficult to stop, especially the "I'm copying it for a friend" idiot. Standing on a soap box in Hyde Park isn't going to do much, so you'll have to take some action. Stopping piracy in markets, clubs and schools is reasonably easy, just contact whoever

runs the market or clubhouse, or point out to the headmaster (or whatever) that illegal activities are taking place under their charge. Most will stop the piracy themselves, especially if you mention that F.A.S.T will be quite interested in visiting them with their solicitors. This method has been successfully used in a number of situations as most heads of clubs, markets and schools would rather not have illegal things going on.

If you notice casual piracy going on in a club or school and it isn't able to be caught by the above method, then you are just going to have to get a little more sneaky....

■ The Oops I've tripped over a lead method:

1) Find suitable power lead (preferably trailing on floor)

2) Trip over it, pulling it out of the computer.

3) say "Oops I've just tripped over this lead"

It is amazing how dropped pens/rulers/wallets etc can cause you to accidentally pull leads out when picking them up. This is good for when leads are in hard to reach places.

■ The Oops, Sorry, wrong switch method:

1) Find suitable wall socket (usually with 2 plugs)

2) Switch off Pirate's computer

3) say "Oh, Sorry, wrong switch"

■ The Sorry, I knocked your disk off the table and trod on it method:

Quite obvious this one, but practise at home with dummy disks (pieces of cardboard) so that you can make it look accidental. Obviously, only do this to destroy the pirated copy, or the copy program, don't destroy the original (un-

less it too is a pirate) and make sure they are not just copying P.D software first.

There are a methods that are even more sneaky, but these are only suitable for use on definite software pirates, and not suggested for use in clubs etc, where innocent disks may be damaged.

■ Fun With Magnets.

It's amazing what a mess a magnet will make of the data on a disk. Accidentally leaving a magnet (or something magnetised) in a pirates disk box will make life interesting.

In no way should you ever use viruses to try and fight piracy, they cause far more trouble than they are worth, and usually backfire. The same techniques as used above, can be used on virus designers by the way!

If you intend to try any of the above, make sure that you won't damage any equipment in any way, and that you won't damage any legally bought software, we're the good guys (and gals) remember. Go get em!

By the way, this article is dedicated to Special Agent Z in Cornwall, and everyone who has made life difficult for pirates.

Useful Phone Numbers

Federation Against Software Theft

0628 660377

ELSPA (European Leisure Software Publishers Association)

0386 930642

Rainbow Warrior Fixes

By Rog. G Ayres

This fixes the problem in Rainbow Warrior where the cursor is about 1/2 an inch below the line being processed: Load Rainbow Warrior into the editor and make the following changes. The line numbers show you which lines to edit:

133) Rain (0 , Y1 + 2) = IDX becomes

Rain(0 , Y1 + 14) = IDX

137) Rain (0 , Y1 + 2) = BC : View

becomes Rain (0 , Y1 + 14) = BC :

View

276) CL = Rain (0 , Y + 2) becomes

CL = Rain (0 , Y + 14)

Now find **procedure GROWBOX** on line 196 and Unfold It, add the following statements to the end of line 215:

GY1 = GY1 + 14 : GY2 = GY2 + 14

Don't forget to save it! It's best to put it on a different disk or use a different name, just in case you've made a mistake with the update.

■ Extra Functions

Rainbow Warrior allows you to do nice metallic bars by using the "n" key to go down a line and set it to the next darker colour. the "N" key does the same thing going up the screen. The change below adds the "o" and "O" keys, but getting brighter rather than darker, and adds the "(" and ")" keys on the numeric keypad to go up or down without changing the colour.

I added this because I found it frustrating to continually switch between keyboard and mouse. The change is in two parts:

1) Replace the old lines 84 and 85

(Beginning If IS="n"...) with the 6 lines below

If IS = " n " Then Gosub FALL :

Gosub DARK

If IS = " N " Then Gosub RISE :

Gosub DARK

If IS = " o " Then Gosub FALL :

Gosub BRIGHT

If IS = " O " Then Gosub RISE :

Gosub BRIGHT

If IS = " (" Then Gosub FALL

If IS = ") " Then Gosub RISE

2) Find the label **DARK:** (old line 139) and insert above it:

FALL: K = 1 : Y = Y Mouse :

Gosub MBIT

YUP = Y Mouse : Inc YUP : Ymouse =

YUP : Return

RISE: K = 1 : Y = Y Mouse :

Gosub MBIT

YUP = Y Mouse : Dec YUP : Ymouse =

YUP : Return

I hope this information is of interest to other club members. I am writing separately to the author "Spadge". I hope he won't mind that I've tweaked his excellent program!

Rainbow Warrior is on APD 76, and is recommended by the AMOS Club for designing Rainbows to brighten up your programs. You can get it from Deja Vu P.D for just £1.75, making it one of the cheapest "Must Get" add ons for AMOS.

Rainbow Warrior allows you to design rainbows to use to either beef up backgrounds in games, or to do "Bar" effects in demos. You simply draw the colour pattern you want, and Rainbow Warrior saves out an ASCII file which you can merge into your program.

Rainbow warrior is used by several of the top AMOS programming teams, including Sparx, Shadow Software and Peter Hickman. Get it!

TEN LINERS

Place Invaders

This 10-liner is what happens when a programmer gets bored waiting for the Japanese Grand Prix to be shown live at 4:50 in the morning.

I wasn't sure it was possible to write a Space Invader clone in 10 lines, so I gave it a go. It took about an hour and a half to write and isn't a perfect clone, but it's good fun, and shows an easy way of getting 50 invading aliens to move about the screen without using more than 3 bobs!

You'll need some bobs for this game, basically 2 animations of 3 different invaders (I cheated and just re-coloured the same one!), as well as a ship for the player, and a bullet. The invaders should be designed as 16x16 pixels in size (bobs 1-6) the gun as 16x16 (Bob 7), and the bullet as 2 2x8 pixel bobs (Bobs 8 & 9).

I've listed this one using the new checksum system, so don't type in the Checksums in the square brackets at the end of the line, or the line number (and round bracket) at the start of the line.

Packer

This one is a simple version of the old Frogger game, i.e. get across a pair of busy roads without ending up as pizza topping.

Controls are simple, Left, Right and Up (you can't go back!). Packer uses two banks of sprites from Sprite 600 on the Extras disk, so make sure that it is in an available drive (or put it in the drive when

prompted if using a single drive).

There aren't any frog sprites in sprite 600, so I compromised and used the pac-man (Allens/Monster1.abk) as the player sprite.

You get 5 Pac People, who you have to get across 4 lanes of traffic. Have Fun!

Again, this one is listed using the new checksum system.

Ann A Conda

This one is based upon the old Snake Byte game, and generate its own very basic graphics. You simply have to collect the green squares (apples) while avoiding the blue squares and your own tail. Compile this one for best results

Win a Compiler or AMOS TOME

We've nearly got enough 10-liners for a Licenseware diskful now, but we still need more (We want around 30-40 on the disk), so keep sending them in.

The best will receive a copy of either the Compiler or AMOS TOME.

The usual 10-liner rules apply. The program must be sent in as an AMOS program on disk (NOT COMPILED!). It can only use up to 10K of external files, with no machine code.

Entries must be received by the end of January 1992. Once the disk is finished, it will be released through Deja Vu Software as licenseware, and the royalties will be split between the programmers of the games. Obviously, with 70p royalty per disk being split between 30 or 40 games, each game's royalty will be very small, so write as many as you can. We can always do a second disk!

★ PLACE ★ INVADERS

- 1) LIVES = 3 : While LIVES > 0 : COUNT = 0 : Screen Open 0 , 320 , 200 , 16 , Lowres : Curs Off : Flash Off : Cls 0 : Screen Open 1 , 320 , 256 , 16 , Lowres : Curs Off : Flash Off : Cls 0 [19]
- 2) Screen Hide 1 : Screen 1 : For R = 0 To 4 : For A = 0 To 9 : Paste Bob A * 16 + 16 , R * 16 + 16 , 2 * (R / 2) + 1 : Paste Bob A * 16 + 16 , R * 16 + 128 , 2 * (R / 2) + 2 : Next A : Next R [8]
- 3) Screen 0 : Get Sprite Palette : Double Buffer : DX = 1 : While LIVES > 0 and COUNT < 50 : DEAD = 0 : X = 0 : EY = 199 : SY = - 16 : Bob Update Off : While DEAD = 0 and COUNT < 50 [57]
- 4) Screen 0 : Bob Clear : Screen Copy 1 , 0 , T * 112 , 192 , 112 + T * 112 To Logic(0) , AX , AY : Bob 1 , X , 199 , 7 : Bob 2 , SX , SY , 8 + T : Bob 3 , EX , EY , 8 + T : Bob Draw : Screen Swap : Wait Vbl [108]
- 5) X = Max (0 , Min (320 , X + (Jleft (1) - Jright (1)) * 2)) : Add T2 , 1 , 0 To 3 : T = T2 / 2 : AX = AX + DX : If AX > = 128 or AX < = 0 : DX = - DX : AY = AY + 8 : End If [47]
- 6) If Fire (1) and SY < = 0 : SY = 170 : SX = X : Shoot : End If : If SY > - 16 : SY = SY - 4 : If Point (SX , SY - 1) and SY > 0 : HX = Max (0 , (SX - AX - 16) / 16) * 16 : HY = Min (4 , Max (0 , (SY - AY - 16) / 16)) * 16 [86]
- 7) Screen 1 : If Point (HX + 24 , HY + 17) > 0 : Cls 0 , HX + 16 , HY + 16 To HX + 32 , HY + 32 : Cls 0 , HX + 16 , HY + 128 To HX + 32 , HY + 144 : SCORE = SCORE + (5 - HY / 16) : Boom : SY = - 16 : Inc COUNT : End If : End If : End If [51]
- 8) Add EY , 2 : If Bob Col (3) and Col (1) : DEAD = 1 : Boom : End If : If EY > 190 : EX = Rnd (160) + AX + 16 : EY = AY : End If : Screen 0 : If Point (X , 182) > 0 : Boom : DEAD = 1 : End If : Wend [81]
- 9) If DEAD : Dec LIVES : End If : Wend : Screen 0 : Paper 0 : Pen 1 : If DEAD : Locate 0 , 10 : Centre "GAME OVER" : End If : Locate 0 , 12 : Centre "Score:" + Str\$ (SCORE) [76]
- 10) If COUNT = 50 : Locate 0 , 14 : Centre "Wave Complete!" : End If : Wait Key : If DEAD = 0 : BY = BY + 16 : AY = Min (48 , BY) : End If : AX = 0 : Wend [73]

PACKER



- 1) Dim TX(11), TS(11) : Dir\$ = "Extras : Sprite_600" : Load "Vehicle / Lorries .
abk" : Load "Allens / Monster1 . abk" , 1 : Hide On : SP = 8 [39]
- 2) Do : Curs Off : Flash Off : Get Sprite Palette : Cls 3 : Ink 9 : For A= 0 To
319 : Draw A , 0 To A , Rnd (8) + 8 : Draw A , 82 - Rnd (8) To A , 108 +
Rnd (8) : Draw A , 199 To A , 191 - Rnd (8) : Next A [12]
- 3) Double Buffer : LIVES = 5 : HME = 0 : DEAD = 0 : While LIVES > 0 and HME <
5 and DEAD = 0 : X = 160 : Y = 184 : For A = 0 To 11 : TX (A) = (A mod 3
) * 144 + Rnd (80) [106]
- 4) TS (A) = A + 1 + ((A / 3) mod 2) * 32768 : Next A : Bob Off : A = 0 :
While A < HME : Paste Bob A * 32 , 1 , 21 : Inc A : Wend : A = 0 : While A <
LIVES : Bob 13 + A , A * 32 , 184 , 23 : Inc A : Wend : Wait Vbl [2]
- 5) While OK = 0 : Bob 1 , X , Y , R + 21 + DEAD : For A = 0 To 11 : Bob 2 + A
, TX (A) , (A / 6) * 40 + (A / 3) * 32 + 24 , TS (A) : Next A : Wait
Vbl [69]
- 6) For A = 0 To 11 : LDX = (SP - A / 3) * ((A / 3) mod 2) * 2 - 1 : TX
(A) = TX (A) + LDX : If TX (A) < - 80 : TX (A) = 320 + Rnd (60) :
End If : If TX (A) > 380 : TX (A) = - 40 - Rnd (40) : End If : Next A
[47]
- 7) R = R + DEAD : OK = R = 12 : If Bob Col (1 , 2 To 12) and DEAD = 0 :
DEAD = 1 : R = 8 : End If : Y = Y + DY : X = X + DX : If DEAD = 0 and (DX
< > 0 or DY < > 0) : R = (R + 1) mod 3 : X = Max(16 , Min (304 , X + DX)
) : Y = Y + DY : End If [65]
- 8) If R = 0 : DX = 4 * (Jleft (1) - Jright (1)) : DY = 4 * (- Jup (1)) * (DX = 0) : End If : If Y < 1 and DEAD = 0 : Inc HME : Inc SP : OK = 1 : End If :
Wend : DX = 0 : DY = 0 : R = 0 : Dec LIVES : DEAD = 0 : OK = 0 : Wend [68]
- 9) Paper 9 : Pen 15 : Locate 0 , 12 : Centre Str\$ (HME) + " Saved !" : If HME =
5 : Locate 0 , 12 : Centre "Perfect Score !" : End If : Wait Key [46]
- 10) Screen Close 0 : Screen Open 0 , 320 , 200 , 16 , Lowres : Loop [74]

ANNOA CONDA

- 1) Dim X (30) , Y (30) : Curs Off : Flash Off : Cls 0 : For A = 0 To 9 : Ink
A + 3 : Bar 0 , 0 To 3 , 3 : Get Bob 1 + A , 0 , 0 To 16 , 4 : Hot Spot 1 + A
, 2 , 2 : Next A : Cls 0 [80]
- 2) Dim EX (6) , EY (6) , BX (10) , BY (10) : Double Buffer : LIVES = 3 :
SCORE = 0 : Ink 3 : Box 0 , 0 To 319 , 199 : Box 1 , 1 To 318 , 198 : While
LIVES > 0 : DEAD = 0 : L = 4 : EL = 0 : NB = 0 : For A = 0 To L : X (A) =
160 + A * 5 : Y (A) = 100 : Next A [27]
- 3) DX = - 1 : DY = 0 : L2 = 0 : While DEAD = 0 : For A = 0 To L : Bob 1 + A ,
X (A) , Y (A) , 2 + (A < 2) + A / 4 : Next A : For A = 0 To EL : Bob 40
+ A , EX (A) , EY (A) , 4 : Next A : For A = 0 To NB : Bob 50 + A , BX
(A) , BY (A) , 3 : Next A : Wait Vbl [123]
- 4) X (0) = X (0) + DX : Y (0) = Y (0) + DY : XL = X (L) : YL = Y (L) :
Add F , 1 , 0 To 3 : If F = 0 : For A = L To 1 Step - 1 : X (A) = X (A - 1
) : Y (A) = Y (A - 1) : Next A : End If [34]
- 5) If F = 0 and L2 > 0 and L < 30 : L2 = 0 : Inc L : X (L) = XL : Y (L) = YL :
End If : If Rnd (100) = 1 : R = Rnd (NB + 1) : BX (R) = Rnd (300) + 10 :
BY (R) = Rnd (190) + 5 : NB = Min (3 , NB + 1) : End If [0]
- 6) If Bob Col (1 , 50 To 59) : For B = 0 To NB : If Col (B + 50) : Inc
SCORE : Inc L2 : BX (B) = Rnd (300) + 10 : BY (B) = Rnd (190) + 5 : End
If : Next B : End If [88]
- 7) If Bob Col (1 , 4 To 49) : DEAD = 1 : End If : If Rnd (100) = 1 : EL = Min (4 , EL + 1) : R = Rnd (EL) : EX (R) = 16 : EY (R) = 4 : End If [36]
- 8) If F = 0 : For A = 0 To EL : EDX = Sgn (X (0) - EX (A)) * 3 : EDY =
Sgn (Y (0) - EY (A)) * 3 : EX (A) = EX (A) + EDX : EY (A) = EY (A) + EDY : Next A : End If [125]
- 9) If F = 0 and Joy (1) : DX = Jleft (1) - Jright (1) : DY = Jup (1) -
Jdown (1) : End If [84]
- 10) If X (0) < 4 or X (0) > 316 or Y (0) < 4 or Y (0) > 196 : DEAD = 1 : End
If : Wend : Bob Off : Boom : Dec LIVES : Wend : Locate 0 , 10 : Paper 0 : Pen
3 : Centre " SCORE " + Str\$ (SCORE * 10) : Wait Key [124]

USING amos TOME

With a significant number of AMOS programmers now using the AMOS TOME Map extension, I thought it would be useful to start a series of articles on using it in your programs.

This issue, I will cover simple vertical scrolling, and how to detect what tile the player is on. We'll be using the tiles and bobs from the Vertigo game on the AMOS TOME disk.

■ Grabbing bobs from a program

As the bobs for Vertigo are saved with the program, you'll need to load the game in (it's called **Vert Scroll.AMOS** on the disk). You can then either save the sprite bank out from direct mode (**Save "Vrtsprites . abk" , 1**) or erase the program by marking it as a block (Ctrl-A) and cutting the block, thus leaving you with a sprite bank and a blank program.

If you have grabbed the sprites by saving them out, NEW the program, and then go to direct mode and load the sprites back in (**Load "Vrtsprites . abk"**).

■ Scrolling a background

The system for updating the background and bobs was explained in Issue 4's Moving Things Behind Things article, so I won't repeat it here.

To make the background scroll needs a few minor modifications. As long as you follow a few general guidelines, you should be able to scroll with no problems. You shouldn't be scrolling the double

buffered screen that you are viewing. It is better to have a hidden (single buffered) background screen and scroll that. It should be opened slightly larger than the area you want to see, to cope with the extra scrolling area. Basically, for every direction that you want to scroll in, you should add the Width or Height of the tiles you will be using to that dimension. For a 4 way (up/down/left/right) scrolling screen, like in the Scroll8 program you need to add twice the tile width to the screen width, and twice the tile height to the screen height. Whereas in the Vertigo game (Which only scrolls downwards), you only need to add the height of the tiles to the screen height.

Every frame, the windowed area is screen copy'ed to the Logical (Hidden) screen of the Double buffered viewing screen. The bobs are placed over it, and it is screen swapped to be seen. The scrolling effect is achieved by screen copying from a different line each frame (rather like moving down the page of a book or newspaper). To find out the line to copy from, we simply use the Y Map Pixel Co-ordinate (MY in this example) and work out how many pixels it is over the current tile co-ordinate (it sounds complex, but it isn't). This is achieved by:
Tile co-ordinate (of the top of the map shown) = MY / Tile Size
Viewing Line = MY mod Tile Size

so we end up with
HY = MY / 32
FY = MY mod 32

We can now use HY to tell us that a new line of tiles needs drawing (if it changes), and FY to show where to screen copy from. This means that the screen copy command we will need is:
Screen Copy 1 , 0 , FY , 256 , 256 + FY to Logic(0) , 0 , 0

When HY changes (if HY < HYO), we

simply scroll screen 1 down by 32 pixels (The height of a tile), and draw the next line of tiles (using Map Top). Note that the scroll is done by using screen copy rather than Def Scroll/ Scroll. This is simply because it is faster and smoother.

■ Spotting Tiles

Another thing that you'll need to do from time to time in map games, is work out what tile you (or an object) are on. There are several ways of doing this, either using **=MAP TILE**, or the more powerful **=TILE VAL**. For this example, we'll use **=MAP TILE** (as there aren't any values assigned to the tiles we are using) to tell us the tile number that the player's ship is flying over.

First you need to work out the tile co-ordinate that the object is at. So you take:

The Map X co-ordinate (MX, in this case 0 as the map is only vertically scrolling)

The Map Y co-ordinate (MY)

The Bobs screen co-ordinates (X & Y)

The Fine Scroll offsets (FX & FY, in this case FX isn't used)

To calculate TX & TY (Tile X & Y), you do
 $TX = (MX + X + FX) / \text{Tile Size}$
 $TY = (MY + Y + FY) / \text{Tile Size}$

In our example, TX is simply $TX = X / 32$ and TY is $TY = (MY + Y + FY) / 32$. You can then get the tile number from this co-ordinate with:

$MT = \text{Map Tile} (TX, TY)$

If you have assigned values to the tiles and have the value list loaded in (We don't use it in this example), you could also do $TV = \text{Tile Val} (TX, TY, \text{list})$ to find out the value of the tile, rather than it's number. In our example, we'll simply zap this tile number and a box (to show which tile we mean) back onto the background.

Time to put this into practice. Get typing! Remember that the line numbers at the start of the lines and the checksums at the end of the lines shouldn't be entered.

- 1) Dir\$ = " TOME UPDATE: " [33]
- 2) Rem load in Tiles and map [11]
- 3) Load " TILES / Verttiles . abk " [19]
- 4) Reserve As Work 6 , 404 [121]
- 5) Bload " MAPS / Vertscroll . map " , Start (6) [28]
- 6) Rem Set tile size and map viewing area [68]
- 7) Tile Size 32 , 32 [61]
- 8) Rem viewing area is set as slightly higher than you will see it [107]
- 9) Map View 0 , 0 To 256 , 288 [73]
- 10) Rem set up screens [66]
- 11) No Icon Mask : Rem stops colour 0 being see through [66]
- 12) Hide On : Auto View Off [118]
- 13) Rem open screen 1 for scrolling area (Hidden) [121]
- 14) Screen Open 1 , 320 , 288 , 16 , Lowres [61]
- 15) Curs Off : Flash Off : Cls 0 [43]
- 16) Screen Hide 1 [61]
- 17) Rem open screen 0 as actual viewing screen [47]
- 18) Screen Open 0 , 320 , 256 , 16 , Lowres [63]
- 19) Curs Off : Flash Off : Cls 0 [43]
- 20) Get Icon Palette : Wait Vbl [105]
- 21) Double Buffer [21]
- 22) Screen To Front 0 [102]
- 23) Auto View On [3]
- 24) HYO = - 1 : Rem force map redraw [9]
- 25) MXY = Map Y * 32 : Rem start at bottom of map [48]
- 26) MY = MXY - 256 : Rem screen is 256 pixels high [43]
- 27) SCR = 1 : Rem scrollspeed [100]

```

28) X = 128 : Y = 128 : T = 0 : Rem co
   - ords of player [ 9]
29) Bob Update Off [ 17]
30) While DEAD < 4 [ 83]
31) HY = MY / 32 [ 54]
32) FY = MY mod 32 [ 81]
33) If HYO <> HY : Gosub VERTMAP :
   End If [ 39]
34) Screen 0 [ 28]
35) Bob Clear [ 54]
36) Rem copy background from
   scrolling screen [ 38]
37) Screen Copy 1, 0, FY, 256, 256
   + FY To Logic ( 0 ), 0, 0 [ 103]
38) HYO = HY [ 82]
39) Rem move map y co - ord [ 22]
40) Add MY, - SCR [ 54]
41) MY = ( MY + MXY ) mod MXY [ 113]
42) Rem display player bob ( T is
   frame toggle ) [ 124]
43) Bob 1, X, Y, 1 + T [ 61]
44) Rem update it, and view it [ 1]
45) Bob Draw [ 111]
46) Screen Swap [ 57]
47) Wait Vbl [ 115]
48) Rem toggle player frame ( 0 / 1 )
   [ 9]
49) T = 1 - T [ 1]
50) Rem move player [ 120]
51) X = Max ( 16, Min ( 240, X + (
   Jleft ( 1 ) - Jright ( 1 ) ) * 8 ) )
   [ 124]
52) Y = Max ( 128, Min ( 240, Y + (
   Jup ( 1 ) - Jdown ( 1 ) ) * 8 ) ) [ 12]
53) If Fire ( 1 ) [ 7]
54) Gosub ANALYSE [ 33]
55) End If [ 96]
56) Wend [ 24]
57) End [ 79]
58) ANALYSE : [ 119]
59) Rem show tile number under player
   [ 21]
60) Rem first work out player's tile co
   - ords [ 23]
61) TX = X / 32 : Rem divide by tile
   size [ 62]

```

```

62) TY = ( MY + Y + FY ) / 32 : Rem
   dlto [ 111]
63) Rem get tile number [ 27]
64) MT = Map Tile ( TX, TY ) [ 64]
65) Rem display tile number and box
   over tile [ 19]
66) Screen 1 [ 61]
67) Ink 1, 0, 1 [ 124]
68) Text TX * 32, ( TY - MY / 32 ) *
   32 + 16, Right$ ( " " + Str$ ( MT )
   , 2 ) [ 63]
69) Box TX * 32, ( TY - MY / 32 ) *
   32 To TX * 32 + 31, ( TY - MY / 32
   ) * 32 + 31 [ 78]
70) Screen 0 [ 60]
71) Return [ 10]
72) VERTMAP : [ 115]
73) Screen 1 [ 61]
74) If HYO = - 1 [ 112]
75) Map Do 0, HY [ 122]
76) Else [ 31]
77) Rem scroll screen 32 pixels ( height
   of tile ) [ 98]
78) Screen Copy 1, 0, 0, 256, 288
   To 1, 0, 32 [ 0]
79) Rem redraw top line [ 40]
80) Map Top 0, HY [ 58]
81) End If [ 64]
82) Return [ 10]

```

A New version of AMOS TOME is being worked on at the moment for release sometime in the new year. Called TOME Series 4, it has features such as having the Tile Maker built into the Map Editor and about another 10-20 effects and commands in the editor. Plus the possibility of new commands in the extension also.

The main plus is that we are also working on a TOME tutorial book which will be part of AMOS TOME System 4. Existing AMOS TOME users will be able to upgrade for a small fee, and will be notified when the new version is available.

DE COLON ISER

The following program will separate your programs into single statement per line, it can also remove rems!

```

1) ' De - Colon - Iser [ 106]
2) ' By Aaron Fothergill [ 99]
3) ' AMOS Club 1991 [ 47]
4) Rem turns multi statement program
   lines into single statement lines [ 15]
5) Screen Open 0, 640, 200, 2,
   Hires : Curs Off : Flash Off : Cls 0 [
   78]
6) Colour 1, $FFF [ 119]
7) Set Input 10, - 1 [ 37]
8) F$ = Fsel$ ( " * . ASC " , " " , "
   Load an ASCII AMOS program " ) [
   47]
9) If F$ <> " " [ 111]
10) If Exist ( F$ ) [ 31]
11) Open In 1, F$ [ 76]
12) OK = 1 [ 40]
13) End If [ 96]
14) End If [ 64]
15) F$ = Fsel$ ( " * . ASC " , "
   NO COLONS . ASC " , " Save it as
   ? " ) [ 49]
16) If F$ <> " " [ 111]
17) Open Out 2, F$ [ 6]
18) OK2 = 1 [ 26]
19) End If [ 64]
20) L = 1 [ 64]
21) Paper 0 : Pen 1 [ 54]
22) While OK = 1 and OK2 = 1 [ 38]
23) Print " Line " ; L [ 8]
24) Inc L [ 40]
25) E = Eof ( 1 ) [ 36]
26) If E = 0 [ 103]
27) Line Input #1, L$ [ 46]
28) Gosub NOCOLON [ 44]
29) Else [ 63]
30) OK = 0 [ 41]
31) End If [ 96]
32) Wend [ 24]
33) Close 2 : Close 1 [ 57]
34) Print " Finished " [ 67]
35) Print COUNT ; " Commands " [ 61]

```

```

36) End [ 79]
37) NOCOLON : [ 122]
38) Rem remove spaces [ 75]
39) While Left$ ( L$ , 1 ) = " " [ 41]
40) L$ = Mid$ ( L$ , 2 ) [ 102]
41) Wend [ 24]
42) Rem to make the program De -
   Rem also, [ 5]
43) Rem remove the rems from before
   the following line [ 5]
44) Rem If ( left$ ( L$ , 3 ) <> " Rem
   " ) and ( left$ ( L$ , 1 ) <> " " )
   [ 97]
45) QU = 0 : Rem flag for quote [ 100]
46) M$ = " " : P = 1 [ 50]
47) While P <= Len ( L$ ) [ 32]
48) If Mid$ ( L$ , P, 2 ) = " : " and
   QU = 0 [ 5]
49) Print M$ [ 24]
50) Print #2, M$ [ 37]
51) Inc COUNT : Inc P [ 9]
52) M$ = " " [ 116]
53) Else [ 63]
54) If QU = 0 [ 38]
55) If ( Mid$ ( L$ , P, 4 ) = " Then "
   ) or ( Mid$ ( L$ , P, 3 ) = " Rem "
   ) [ 88]
56) M$ = M$ + Mid$ ( L$ , P ) [ 103]
57) P = Len ( L$ ) [ 67]
58) End If [ 64]
59) End If [ 96]
60) If Mid$ ( L$ , P, 1 ) = Chr$ ( 34 )
   : Rem quote [ 83]
61) QU = 1 - QU [ 33]
62) End If [ 96]
63) M$ = M$ + Mid$ ( L$ , P, 1 ) [ 90]
64) End If [ 96]
65) Inc P [ 52]
66) Wend [ 24]
67) If M$ <> " " : Print M$ : Print #2,
   M$ : Inc COUNT : End If [ 30]
68) Rem for De - rem, remove Rem
   from the following line [ 23]
69) Rem endif [ 26]
70) Return [ 10]

```

Compiler Bugs

As mentioned in the last issue, several minor (but very annoying) bugs have been found in the AMOS Compiler. Although these are not serious enough to stop use of the Compiler, they can be annoying from time to time. Some bugs, such as Under On becoming Inverse On when compiled, are fixed in V1.32 of AMOS, but the following bugs are still there:

The VAL Bug About 90% of the time, the VAL command won't work once compiled. Symptoms are usually random (very large) numbers or zeros returning from the function and throwing any program relying on the function. The only real way around it (i.e. that works), is to replace the =VAL function with the following routine. Simply do GVAL[\$] and the variable V# will have the value.

```
1) Input " Enter a number " ; N$ [ 13]
2) GVAL [ N$ ] [ 112]
3) Print V# [ 36]
4) End [ 79]
5) Procedure GVAL [ V$ ] [ 37]
6) Shared V# [ 92]
7) V = 0 [ 123]
8) P = 1 [ 124]
9) M = 0 [ 96]
10) DP = Instr ( Flp$ ( V$ ) , " . " ) [ 28]
11) If DP>0 : Dec DP : End If [ 35]
12) DP# = 1 : A = 0 : While A<DP :
    DP# = DP# * 10 : Inc A : Wend [ 80]
13) Repeat [ 55]
14) If ( Mid$ ( V$ , P , 1 ) > " 0 " )
    and ( Mid$ ( V$ , P , 1 ) < " 9 " )
    [ 111]
15) V = V * 10 + Asc ( Mid$ ( V$ , P , 1 ) ) - 48 [ 58]
16) Else [ 63]
17) If Mid$ ( V$ , P , 1 ) = " - " [ 105]
18) M = 1 [ 65]
19) End If [ 64]
20) End If [ 96]
21) Inc P [ 52]
```

```
22) Until P>Len ( V$ ) [ 48]
23) If M : V = - V : End If [ 50]
24) V# = V / DP# [ 38]
25) End Proc [ 65]
```

Another bug is that when compiling to AMOS, the later versions of the compiler add 512K to the required Set Buffer size at the start of the program, giving an Out Of Memory error. To cure this, just subtract 512 from the Set Buffer value at the start of the Compiled program, and save it again.

If you are using AMAL in your compiled program, make sure your AMAL strings end with a semi-colon (;), as although you don't need them at the end of AMAL strings in an interpreted AMOS program, the Compiler tends to hit you with AMAL Errors if you don't.

Also watch out for using Strings in Procedures, as we warned you about in the last issue. The situation is much worse when compiling.

One that had us stumped for a while is that if you do a very fast sequence of random numbers. E.g. For A = 1 to 1000 : B = Rnd(10) : Next A, the compiled program will crash. Simply slow the routine down with a wait command to get around this.

If you have any extensions loaded (such as TOME, CTEXT or 3D), then make sure that they are listed in the RAMOS.ENV file, as this is the file that the compiler uses, rather than AMOS1_3_PAL.ENV or AMOS1_3_NTSC.ENV. The normal symptoms are an Extension Not Present Error when compiling. Just add the extensions to the loaded extension list of RAMOS.ENV in the CONFIG1_3.AMOS program.



LICENSEWARE REVIEWED

Formula 1 Challenge (1 Meg, Disconnect all Drives)

By

LPD 44

As I'm a bit of a motor racing fan, I was looking forward to reviewing this game.

It is a Grand Prix team management game, so that rather than driving the cars, you have to manage the team through the season to try and win the Drivers' and Constructors' championships.

This is a great idea, and the game has some nice touches, but it doesn't have enough going on to be a management simulation, and tends towards the boring rather rapidly.

As for each race, your only decisions are what tires (5 types), what suspension setting (Hard, Med, Soft) and remember to tune up the engine. The rest is down to how good a driver you can afford, and pure luck.

The game doesn't go in for total realism, although this is a good point (who would have thought a couple of years ago that both Lotus and Brabham would both be on the point of bankruptcy), but a Dallara (A VERY slow Grand Prix car) winning 2 races in the 1991 season is possibly a little silly. As I've said, this isn't really a bad thing, as it does make you think what would happen if all the Grand Prix drivers got shuffled into random teams!

This is a game that would be a lot better if it was re-written to be a bit more involving, with more rewards for success and more variety.

Stocking Fillers (1 Meg)

By Sparx

LPD 48

This is a disk of 3 games for children at Christmas, it makes an ideal present for a young child, as the disk customises itself with the child's name when you first boot it up.

The games are all simple to play, with enough variety to keep them happy and away from tying up granny for at least a couple of hours.

Two of the games are Christmas versions of old favourites, being **Christamajig** (A cross between Sparx' Thingamajig, Jigsaw game and Colouring Book games) and **Pair It**, with Christmas picture cards. Both are very straightforward, but great fun. The other game is called **Santa's Sleigh Ride**, and it is a scrolling game, in which Santa must drop presents down chimneys. Very simple, but there are various things going on to make his life difficult, so it will keep a child's attention for some time.

The level of difficulty has been kept low, so that it isn't frustratingly hard to play. Even I can beat it (This is from someone who repeatedly gets thrashed at Flower Power by 10 year olds!)

If you have children, (or are an 8 year old who's parents have just flown off to France, leaving you alone with a credit card!) buy this one as a Christmas present!

Magic Forest II and Wilbert's Christmas Wonderland (1 Meg, Disconnect Drives)

By Shadow Software

LPD 39 and LPD 777

Magic Forest II is the long awaited follow up to the simple platform game that came with AMOS. The new version is

horizontally scrolling, has more levels and more gameplay, and is downright fun!

The aim of the game is simple, zap all the red meanies that are out to get you, and collect all the bonus goodies on the way to meeting your girlfriend at the edge of the forest. By collecting falling apples, you can get to bonus levels to try and gain extra lives. By either shooting the apples, or constantly hitting them, you can get them to explode and kill the meanies (the only way of killing them).

Just in time for Christmas, Adam has finished the Christmas version, Wilbert's Winter Wonderland. This is the same game, but with a completely different set of graphics, levels and music, making it more Christmassy. The gameplay is slightly more difficult, as most of the platforms have ice on them, making them slippery.

Definite buys for any fan of platform games.

X-Stitch (1 Meg)

by Derek Pell

LPD 42

This utility is based around Derek's CLI Screen dump routine (also available on Licenseware), and is intended to convert IFF pictures (low or hires, but not HAM) into Cross Stitch patterns for needlework fanatics.

Having seen the success of similar programs for other computers in America, this one should be popular. The program is simple to use and gives very good results.

All the above Licenseware disks can be bought from Deja Vu Software for £3.50 each. **Deja Vu 0942 495 261**

Deja Vu Software

25 Park Road, Wigan, WN6 7AA

New P.D prices

The Official AMOS P.D Library have dropped their prices for P.D (Not Licenseware) software to £1.75 per disk. As of November 1st 1991, the prices are:

£1.75 per disk U.K (Incl V.A.T)

£2.00 per disk Europe, Eire

£2.25 per disk Rest of World

add 50p postage and package to total order

These prices apply to AMOS P.D, Aussie AMOS P.D, Deja Vu General P.D, Fish Disks and TBAG P.D disks

Licenseware disks are still £3.50 per disk.

As Deja Vu is the official AMOS P.D Library, AMOS P.D Disks are always available from there first, and they are always the latest versions (Some other P.D libraries are still selling the AMOS V1.23 update as the latest version).

■ Deja Vu Top Ten

1 APD 36 : AMOS Updater Disk (Versions 1.23 to 1.32)

2 LPD 3 : Galleons (yay!)

3 LPD33 : AMOS Adventure Toolkit

4= LPD13 : Jigmanla

= LPD35 : T-Tecdraw

6= LPD21 : Quingo

= APD266 : AMOS 3D Demo

8: APD255 : Hanlssls V demo 2

9= LPD31 : AMOS CLI Printer Dump

= APD76: Rainbow Warrior

Win 10 free P.D Disks !

See the resubscription form on the back page for details!

Resubscription Offers

It's resubscription time, details of how to resubscribe are on the back page.

Here are some special offers we are running for the resubscription.

If you order AMOS TOME (at the AMOS Club member's price of £19.99) when you re-subscribe, then you get the AMOS TOME Goodies Disk 1 free.

Also, if you haven't already got the excellent Sprite X and Ctext packages, then you can order them for just 2 pounds each (saving 50p on the normal member's price). Postage and packing is included (even for overseas orders), but both of these offers run out on December 20th 1991, so order now!

■ Win Lots of Software !

All resubscription forms that get to use by the first of January 1992, will be put in a large hat (well, a large bag) and the first drawn out will win a complete AMOS developer's kit, containing:-

The AMOS Compiler (supplied by Europress Software), AMOS TOME V3.1, Sprite X V1.32, CTEXT (Supplied by Shadow Software), Rainbow Warrior (By Spadge, Supplied by Deja Vu P.D) and 10 P.D Disks of the winner's choice (Also supplied by Deja Vu P.D).

2 runners up can choose The AMOS Compiler or AMOS TOME, and there are 4 more prizes of Sprite X and Ctext for the next 4 out of the bag.

Winners will be announced in the next newsletter (Volume 2, Issue 1), which should be ready mid January 1992.

Back Issues

Once Volume 2, Issue 1 of the newsletter is released (to everyone who resubscribes), Issues 1-6 will become available as back issues, and any new subscribers, sending in the old membership fee of £10 (£12 Europe) will receive these back issues with information on how to get the new subscription.

Similarly, anyone subscribing from Volume 2 onwards will be able to purchase the full set of back issues for £10.

As this issue will be the last before Christmas (Volume 2, Issue 1 should be out in mid to late January), I and all the staff at Shadow Software/The AMOS Club would like to wish you all a Merry Christmas and a Happy New Year.

I will actually be here over Christmas, and the helpline will be open, although don't expect me to be at my most brilliant when I'm hung over on New Year's day!

MERRY CHRISTMAS!

(And a Happy New Year!)

This Newsletter was written by and is (C) Aaron Fothergill 1991, unless otherwise stated.

Graphics by and (C) Adam Fothergill 1991.

Printed by Creative Printers, 104 Boutport Street, Barnstaple, N.Devon. Telephone (0271) 77655

RESUBSCRIPTION

It's that time of year folks ! To get the next six issues of the Official AMOS Newsletter, and new membership number, giving you access to the AMOS Helpline, fill in the following resubscription form, and send it off with the correct payment to

**The AMOS Club ,
1 Lower Moor ,
Whiddon Valley ,
Barnstaple,
N . Devon .
E X 3 2 8 N W .
U . K .**

Unfortunately, the subscription cost has had to go up slightly (As VAT will have to be charged on magazines from 1992, and postage costs have gone up again), and the cost for U.K subscription (Including BFPO) is £12.00. All overseas memberships cost £15.00. If you have only just joined (September onwards), then your membership number will allow you to use the helpline until June.

✂

I wish to resubscribe to the AMOS Club

Name.....

Address

Postcode.....

I enclose

- ☐ **£12.00 for U.K membership**
- ☐ **£15.00 for Overseas membership**
- ☐ **£19.99 for AMOS TOME (& free goodies disk)**
- ☐ **£2.00 for AMOS Sprite X V1.32**
- ☐ **£2.00 for AMOS CTEXT**

Total £.....

Overseas Cheques must be made out in British Pounds Sterling, and drawn on a British bank. We can accept American Dollar Cheques, but add \$6 to covert bank charges, and make sure that you make them out at the current rate of exchange (U.S Postal Money Orders made out in pounds sterling are easier). Hopefully, we'll be able to accept credit card orders sometime in the new year.

Remember, the AMOS Club is the ONLY AMOS support group with a helpline, and the ONLY AMOS support group backed by Mandarin/Europress software. Please post early if you want any of these for Christmas.